User Guide for DYCORS Algorithm– MATLAB

Juliane Müller

Cornell University
School of Civil and Environmental Engineering

email: juliane.mueller2901@gmail.com

June 18, 2014

1 Introduction

This user guide accompanies the DYCORS algorithm [2] for global optimization problems. The algorithm attempts to find accurate solutions for minimization problems of the following form:

$$\min f(\mathbf{x}), \qquad \text{subject to } -\infty < x_i^l \leq x_i \leq x_i^u < \infty,\ i = 1, \ldots, d, \tag{1}$$

where $f(\mathbf{x})$ is a computationally expensive objective function (often a time consuming simulation model) whose analytical description is not available (black box). Considered are box-constrained optimization problems, i.e. only lower (x_i^l) and upper (x_i^u) variable bounds exist for $x_i \in \mathbb{R},\ i = 1, \ldots, d$, where d is the problem dimension. There are no other constraints. The major difference between DYCORS and Stochastic RBF (see the codes for StochasticRBF by the same author) is that DYCORS is more suitable for large dimensional problems (> 30 dimensions) since it does not perturb all variables of the best point found so far in order to create candidate points, but rather each variable is perturbed with probability

$$P(n) = p_0 \left[1 - \frac{\log(n - m + 1)}{\log(N_{\max} - m)}\right], \tag{2}$$

for all $m \leq n \leq N_{\max}$, and where m is the number of points in the initial experimental design, $p_0 = \min(1, 20/d)$, n is the iteration number, and $N_{\max}$ is the maximum number of allowed evaluations for the optimization. Hence, the probability of perturbation for each variable decreases as the optimization advances (as n grows). It is ensured that at least one variable is perturbed.

Note that for problems with computationally cheap function evaluations the algorithm may not be very efficient since in that case the computational overhead from the optimization routine itself will be more than the overhead from doing function evaluations. Surrogate models are intended to be used when a single function evaluation takes from several minutes to several hours or more. When reading this manual it is recommended to simultaneously take a look at the code and to try out the examples. It is assumed that the user is familiar with the paper on whose content this implementation is based:

- R.G. Regis and C.A. Shoemaker. Combining Radial Basis Function Surrogates and Dynamic Coordinate Search in High-Dimensional Expensive Black-Box Optimization. Engineering Optimization, Vol. 45, Issue 5, pp. 529-555, 2013.

This paper should be cited and the codes should be acknowledged (giving its link) whenever they are used to generate results for the user's own research. The user is urged to read the paper before continuing with the manual since it helps understanding the following descriptions.

The author of this Matlab implementation is

- J. Müller, `juliane.mueller2901@gmail.com`

This implementation contains the option for doing several function evaluations in parallel (in addition to the option of doing one evaluation at a time).

The code is set up such that the user only has to define his/her optimization problem in a Matlab file (see Section 6.1). Additional input such as the maximum number of allowed function evaluations, the number of trials, an indication if the results should be plotted, and the number of function evaluations to be done in every iteration are optional, and if not given by the user, default values are assigned (see Section 6).

This document is structured as follows. In Section 2 the general surrogate model algorithm is described. The installation of the algorithm is described in Section 3. The dependencies of the single functions in the code are shown in Section 4. Section 5 briefly describes the main function DYCORS.m. Section 6 describes the options for the input arguments of the main function and contains an example. Further examples for using the DYCORS algorithm are given in Section 7. The elements of the saved results are described in Section 8.

Finally, if you have any questions and recommendations, or if you encounter any bugs, please feel free to contact me at the email address `juliane.mueller2901@gmail.com`.

2 Surrogate Model Algorithms

Surrogate models (also known as response surfaces or metamodels) are used in optimization algorithms to approximate expensive simulation models [1]. During the optimization phase information from the surrogate model is used in order to guide the search for improved solutions. Using the surrogate model instead of the true simulation model reduces the computation time considerably. Most surrogate model algorithms consist of the same steps as shown in the algorithm below.

Algorithm ***General Surrogate Model Algorithm***

1. Generate an initial experimental design.
2. Do the costly function evaluations at the points generated in Step 1.
3. Fit a response surface to the data generated in Steps 1 and 2.
4. Use the response surface to predict the objective function values at unsampled points in the variable domain to decide where to do the next expensive function evaluation.
5. Do the expensive function evaluation at the point(s) selected in Step 4.
6. Use the new data to update the surrogate model.
7. Iterate through Steps 4 to 6 until the stopping criterion has been met.

Typically used stopping criteria are a maximum number of allowed function evaluations (adopted in this implementation), a maximum allowed CPU time, or a maximum number of failed iterative improvement trials.

3 Installation

Download the file DYCORS.zip and unzip it in a location known to the Matlab search path. Alternatively, you can add a new folder to the Matlab search path by clicking in the Matlab window on

File → Set Path... → Add with Subfolders → Save

You can try if the algorithm works by typing

```
>> DYCORS('datainput_hartman3',200,3,1,1)
```

into the command prompt.

4 Code Structure

The structure of the code is outlined here. The function at the highest level (DYCORS.m) is the main function that has to be called by the user. The subtrees indicate dependencies between the subfunctions.

```
DYCORS.m
└── DYCORSrestartManager.m
    └── DYCORSrestart.m
        ├── SLHDstandard.m
        └── DYCORS_opt.m
            ├── InitialRBFMatrices.m
            │   └── phi.m
            ├── Minimize_Merit_Function.m
            │   └── ComputeRBF.m
            │       └── phi.m
            └── phi.m
```

5 The Main File DYCORS.m

The file from which to run the algorithm is DYCORS.m. The file expects several inputs (see Section 6) of which only the first one is mandatory. The algorithm starts by checking if the input is correct and assigns default values to variables that have not been set by the user. If any mandatory input data is missing or incorrect, the algorithm terminates with an error message indicating where the problem may be. Parameters, such as the type of the used RBF model, the corresponding polynomial tail, and the number of candidate points, are set. The algorithm then calls DYCORSrestartManager.m. After the optimization finished, a plot of the results is generated (if so desired by the user). The algorithm saves the results in the file Results.mat.

6 Input

The main file DYCORS.m requires several input arguments (see Table 1), out of which only the first argument is mandatory to run the algorithm. If no input is given for the remaining arguments, default values are used.

Table 1: Input parameters

Input	Description
data_file	string with name of file containing optimization problem data (mandatory!)
maxeval	positive integer defining maximum number of allowed function evaluations (default $20 \cdot d$, d =dimension), has to be larger than $2(d+1)$
Ntrials	positive integer defining the number of times the algorithm is executed for the given problem (default 1)
PlotResult	0 = no plot; 1 = plot (default 1)
NumberNewSamples	positive integer defining the number of points selected in every iteration of the algorithm for doing expensive simulation (default 1)

6.1 Input data_file

The data file contains all the necessary problem information. See for example the file datainput_hartman3.m. The data file has no input argument, and one output argument (the structure variable Data). The Data structure must contain the information shown on Table 2. It is recommended to scale the variable domain such that Data.xlow = 0 and Data.xup = 1. See, for example, the file datainput_Branin.m for how to do it.

Table 2: Contents data file

Variable	Description
Data.xlow	variable lower bounds, row vector with d (=dimension) entries
Data.xup	variable upper bounds, row vector with d (=dimension) entries
Data.dim	problem dimension, positive integer
Data.objfunction	handle to objective function/simulation model, must return a scalar value

6.2 Input Ntrials

The input Ntrials indicates how often DYCORS.m should be run for the same problem. The reason for running the algorithm more than once for the same problem is the random component when creating the initial experimental design and when generating candidate points. In order to average out the effect of these random components, several trials should be made. However, for computationally expensive problems this might not be possible due to the required computation time for doing the expensive function evaluations. Hence, for most application problems, Ntrials =1 is a reasonable choice.

6.3 Input PlotResult

If set to 1 (or any value different from 0), a plot of the best objective function value averaged over all trials after a given number of function evaluations is made. This allows the user to see the progress of the algorithm and assess convergence.

6.4 Input NumberNewSamples

The variable NumberNewSamples indicates how many points are to be selected in every iteration of the algorithm for doing expensive function evaluations. If NumberNewSamples is larger than one, then the function evaluations are done in parallel. Therefore, Matlab's Parallel Computing Toolbox must be installed. The objective function values for the points in the initial experimental design are in this implementation computed iteratively such that users who do not have Matlab's Parallel Computing Toolbox are able to execute the serial version. If the user has the Matlab Parallel Computing Toolbox installed and wishes to do the function evaluations of the points in the initial experimental design in parallel, go to the file DYCORS_opt.m, comment out the commands

```
%for serial evaluation of points in initial starting design:
%----------------------- SERIAL -----------------------------------------
for ii = 1:Data.m %go through all Data.m points
    time1 = tic; %start timer for recording function evaluation time
    Data.Y(ii,1) = feval(Data.objfunction,Data.S(ii,:)); %expensive simulation
    Data.fevaltime(ii) = toc(time1); %record time for expensive evaluation
    if ii == 1 %initialize best point found so far = first evaluated point
        Data.xbest=Data.S(ii,:); %best point
        Data.Fbest=Data.Y(ii); %best objective function value
    else %update best point found so far if necessary
        if Data.Y(ii) < Data.Fbest
            Data.Fbest=Data.Y(ii); %best objective function value
```

```
            Data.xbest=Data.S(ii,:); %best point
        end
    end
end
%-------------------- END SERIAL ----------------------------------------
```

and uncomment the lines

```
% m=Data.m;
% Y=Data.Y;
% S=Data.S;
% ObjFunction=Data.objfunction;
% Time=Data.fevaltime;
%
% parfor ii = 1:m %go through all m points
%     time1 = tic; %start timer for recording function evaluation time
%     Y(ii,1) = feval(ObjFunction,S(ii,:)); %expensive simulation
%     Time(ii) = toc(time1); %record time for expensive evaluation
% end
%
% [Data.Fbest, IDfbest]=min(Y(1:m));
% Data.xbest=S(IDfbest,:); %best point
%
% Data.Y=Y;
% Data.fevaltime=Time;
```

6.5 Input Example

The following example executes the DYCORS algorithm for finding the minimum of the three-dimensional Hartmann function defined in the file datainput_hartman3.m. The maximum number of function evaluations is set to 300, Ntrials is set to 5 (the algorithm is started 5 times for the problem, and each trial has a different seed for the random number generator). PlotResult is set to 1 in order to illustrate the development of the objective function value vs. the number of function evaluations, and NumberNewSamples is set to 4, i.e. in every iteration four new points are selected and the objective function values of these four points are computed simultaneously using Matlab's parallel for loop (parfor). The user is encouraged to try out the example by typing into the Matlab command window (make sure the location of the files is known to Matlab's search path):

```
>> DYCORS('datainput_hartman3',300,5,1,4)
```

Note that in the command window the iteration number and the number of function evaluations done so far is shown. The plot of the average objective function value vs. the number of function evaluations should look similar to the graph in Figure 1.

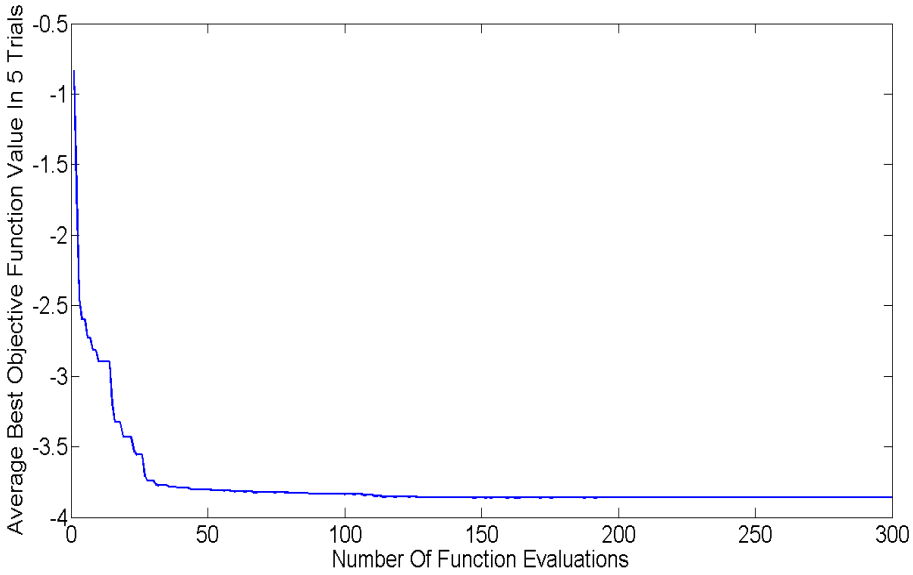

Figure 1: Average objective function value vs. number of function evaluations.

7 Examples

This section contains test problems for continuous black-box optimization problems. The input data files are included in the zip-archive, in the folder ExampleContinuous. The examples have computationally cheap objective functions in order to reduce the computation time when experimenting with the code.

- datainput_Ackley30.m
- datainput_Ackley200.m
- datainput_Branin.m
- datainput_hartman3.m

For finding an approximation of the minimum of the 30-dimensional Ackley function, type

```
>> DYCORS('datainput_Ackley30',400, 1, 1, 5);
```

With this input the following settings are used (in the sequence of input arguments):

- datainput_Ackley30 is the data file
- 400 function evaluations are allowed
- the algorithm is executed one time (=1 trial)
- the results are illustrated in a plot
- 5 points are evaluated in every iteration

8 Results

The algorithm saves the results of the optimization to the file Results.mat. Type

```
>> load Results.mat
```

into the command window to load the data. If the algorithm has not been interrupted (e.g. by pressing CTRL+C), the following elements are contained in the saved Solution structure (see Table 3).

Table 3: Saved Solution structure elements

Elements	Description
Solution.BestPoints	(Ntrials×d) matrix with best point found in each trial of the algorithm
Solution.BestValues	(Ntrials×1) matrix with best objective function value found in each trial of the algorithm
Solution.NumFuncEval	(Ntrials×1) matrix with number of function evaluation in each trial
Solution.AvgFuncEvalTime	(Ntrials×1) matrix with average time needed for evaluating the objective function in each trial
Solution.FuncVal	(maxeval × Ntrials) matrix with objective function values in every trial (ith column corresponds to ith trial)
Solution.DMatrix	(maxeval ×d× Ntrials) matrix with points where objective function has been evaluated in each trial. Third dimension corresponds to trial number
Solution.NumberOfRestarts	(Ntrials × 1) matrix with number of optimization restarts in each trial. The optimization reboots whenever a local optimum has been encountered and if there is a budget of function evaluations left.

9 GNU Free Documentation License

References

[1] A.J. Booker, J.E. Dennis Jr, P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Multidisciplinary Optimization*, 17:1–13, 1999.

[2] R.G. Regis and C.A. Shoemaker. Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, 45:529–555, 2013.